Digital Switching in the Quantum Domain

I.M. Tsai* and S.Y. Kuo†

Department of Electrical Engineering,
National Taiwan University,
Taipei, Taiwan.

Abstract

In this paper, we present an architecture and implementation algorithm such that digital data can be switched in the quantum domain. First we define the connection digraph which can be used to describe the behavior of a switch at a given time, then we show how a connection digraph can be implemented using elementary quantum gates. The proposed mechanism supports unicasting as well as multicasting, and is strict-sense non-blocking. It can be applied to perform either circuit switching or packet switching. Compared with a traditional space or time domain switch, the proposed switching mechanism is more scalable. Assuming an $n \times n$ quantum switch, the space consumption grows linearly, *i.e.* $O(n)$, while the time complexity is $O(1)$ for unicasting, and $O(\log_2 n)$ for multicasting. Based on these advantages, a high throughput switching device can be built simply by increasing the number of I/O ports.

1 Introduction

The demand for bandwidth is rapidly increasing due to the explosive growth of network traffic. Networking technologies play an important role in bridging the gap between limited resources and the constantly increasing demand. In order to avoid a full mesh architecture, a switching device is required to build a realistic network. Over the past few years, a lot of enabling technologies have emerged as candidates for achieving high performance switching. Basically, switches act like automated patch-panels, switching all the electrical or optical signals from one port to another. Traditionally, digital switching can be done in many ways. For example, by allocating physical separated paths, switching can be done in the space domain. A 2-D MEMS optical switch with precisely controlled

*E-mail : tsai@lion.ee.ntu.edu.tw
†E-mail : sykuo@cc.ee.ntu.edu.tw

micromirrors is essentially a space domain switch. Similarly, by associating the data from each port with a unique resource, switching can be performed in many other ways, such as in the time domain, the wavelength domain, and even a combination of these mechanisms.

On the other hand, quantum information science is a relatively new field of study. Quantum computers were first discussed in the early 1980's [1],[2],[3]. Since then, a great deal of research has been focused on this topic. Remarkable progress has been made due to the discovery of secure key distribution [6], polynomial time prime factorization [4], and fast database search algorithm [5]. These results have recently made quantum information science the most rapidly expanding research field. Other applications, such as clock synchronization [7],[8], and quantum boolean circuit implementation [9] have driven this field further into the phase of real-world applications.

In this paper, we present a architecture and implementation algorithm such that digital data can be switched in the quantum domain. First we define the connection digraph which can be used to describe the behavior of a switch at a given time, then we show how a connection digraph can be implemented using elementary quantum gates. The proposed mechanism supports unicasting as well as multicasting and is strict-sense non-blocking [10]. It can be applied to perform either circuit switching or packet switching. Compared with a traditional space or time switch, the proposed switching mechanism is more scalable. Assuming an $n \times n$ quantum switch, the space consumption grows linearly, *i.e.* $O(n)$, while the time complexity is $O(1)$ for unicasting and $O(\log_2 n)$ for multicasting. Based on these advantages, a high throughput switching device can be built simply by increasing the number of I/O ports.

2 Notations and Preliminaries

2.1 Quantum State and Quantum Gates

In a two-state quantum system, each bit can be represented using a basis consisting of two eigenstates, denoted by $|0\rangle$ and $|1\rangle$ respectively. These states can be either spin states of a particle ($|0\rangle$ for spin-up and $|1\rangle$ for spin-down) or energy levels in an atom ($|0\rangle$ for ground state and $|1\rangle$ for excited state). These two states can be used to simulate the classical binary logic.

A classical binary logic value must be either **ON** (1) or **OFF** (0), but not both at the same time. However, a bit in a quantum system can be any linear combination of these two states, so we have the state $|\psi\rangle$ of a bit as

$$|\psi\rangle = c_0|0\rangle + c_1|1\rangle, \tag{1}$$

where c_0, c_1 are complex numbers and $|c_0|^2+|c_1|^2 = 1$. In column matrices, this is written as

$$|\psi\rangle = \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} . \tag{2}$$

The state shown above exhibits an unique phenomenon in quantum mechanics called *superposition*. When a particle is in such a superposed state, it has a part corresponding to $|0\rangle$ and a part corresponding to $|1\rangle$, at the same time. When you measure the particle, the system is projected to one of its basis (*i.e.* either $|0\rangle$ or $|1\rangle$). The overall probability for each state is given by the absolute square of its amplitude. Taking the state $|\psi\rangle$ in Eq.(1) as an example, the coefficient $|c_0|^2$ and $|c_1|^2$ represents the probability of obtaining $|0\rangle$ and $|1\rangle$ respectively. Obviously, the sum of $|c_0|^2$ and $|c_1|^2$ will be 1 to satisfy the probability rule. To distinguish the above system from the classical binary logic, a bit in a quantum system is referred to as a quantum bit, or *qubit*.

Two or more qubits can also form a quantum system jointly. A two-qubit system is spanned by the basis of the tensor product of their own spaces. Hence, the joint state of qubit A and qubit B is spanned by $|00\rangle_{AB}$, $|01\rangle_{AB}$, $|10\rangle_{AB}$, and $|11\rangle_{AB}$, *i.e.*

$$|\phi\rangle_{AB} = c_0|00\rangle_{AB} + c_1|01\rangle_{AB} + c_2|10\rangle_{AB} + c_3|11\rangle_{AB}, \tag{3}$$

where c_0, c_1, c_2, c_3 are all complex numbers and $|c_0|^2 + |c_1|^2 + |c_2|^0 + |c_3|^2 = 1$. In matrix form, this is equivalent to

$$|\phi\rangle_{AB} = \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} . \tag{4}$$

The notations described above can be generalized to multiple-qubit systems. For example, in a three-qubit system, the space is spanned by a basis consisting of eight elements ($|000\rangle_{ABC}$, $|001\rangle_{ABC}$, ..., $|111\rangle_{ABC}$).

A quantum system can be manipulated in many different ways, called *quantum gates*. A quantum gate can be represented in the form of a matrix operation. For example, a quantum *'Not'* (**N**) gate applied on a single qubit can be represented by multiplying a 2×2 matrix

$$N = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \tag{5}$$

which changes the state from $|1\rangle$ to $|0\rangle$ and from $|0\rangle$ to $|1\rangle$, as

$$N \cdot \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} c_1 \\ c_0 \end{pmatrix} . \tag{6}$$

The symbol of an **N** gate is shown in Fig.1(a). Note that the horizontal line connecting the input and the output is not a physical wire as in classical circuits, it represents a qubit under time evolution.

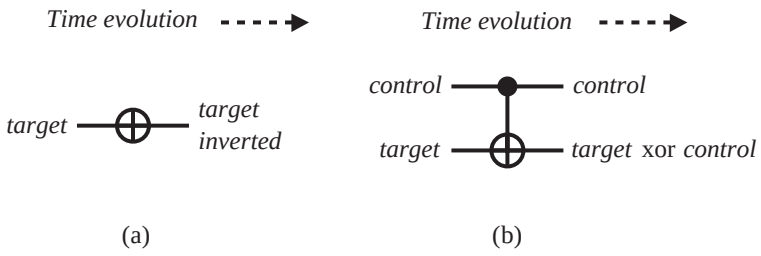

Figure 1: The symbol and bit-wise operation for **N** and **CN** gate.

Similarly, a two-bit gate can be represented by a 4×4 matrix. For example, a *'Control-Not'* (**CN**) gate is represented by

$$CN = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} . \tag{7}$$

The symbol of a **CN** gate is shown in Fig.1(b). A **CN** gate consists of one *control* bit x, which does not change its value, and a *target* bit y, which changes its value only if $x = 1$. Assuming the first bit is the control bit, the gate can be written as $CN(|x, y\rangle) = |x, x \oplus y\rangle$, where '$\oplus$' denotes exclusive-or. In matrix form, a **CN** gate changes the probability amplitudes of a quantum system as follows:

$$CN \cdot \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} c_0 \\ c_1 \\ c_3 \\ c_2 \end{pmatrix} . \tag{8}$$

Further generalization of the quantum gates described above involves *rotation* and *phase shift*. They control the phase difference and relative contributions of the eigenstates to the whole state. For example, a general single bit operation can be represented using a matrix

$$U = \begin{pmatrix} e^{i(\delta+\frac{\alpha}{2}+\frac{\beta}{2})}\cos(\frac{\theta}{2}) & e^{i(\delta+\frac{\alpha}{2}-\frac{\beta}{2})}\sin(\frac{\theta}{2}) \\ -e^{i(\delta-\frac{\alpha}{2}+\frac{\beta}{2})}\cos(\frac{\theta}{2}) & e^{i(\delta-\frac{\alpha}{2}-\frac{\beta}{2})}\sin(\frac{\theta}{2}) \end{pmatrix} . \tag{9}$$

This matrix can also be used to control the change between any two probability amplitude components in a quantum system. Note that, to satisfy the probability rule, all quantum gates U in their matrix form are unitary, *i.e.*

$$UU^{\dagger} = I, \tag{10}$$

where $U^\dagger$ is the conjugate transpose of U.

Just like **AND** and **NOT** form a universal set for classical boolean circuits, one- and two-bit gates are sufficient to implement any unitary operation [11], [12]. A set of quantum gates which can be used to implement any unitary operation is called a universal set. There are many universal sets of one- and two-bit gates. A practical approach is to use general one-bit rotation gates as in Eq.(9) and the **CN** gate as a universal set.

2.2 Qubit Permutation and Replication

An important property regarding a quantum boolean operation is that any quantum boolean logic can be represented using a *permutation*. A permutation is a one-to-one and onto mapping from a finite order set onto itself. A typical permutation P is represented using the symbol

$$P = \begin{pmatrix} a & b & c & d & e & f \\ d & e & c & a & f & b \end{pmatrix} . \tag{11}$$

This permutation changes $a \rightarrow d$, $d \rightarrow a$, $b \rightarrow e$, $e \rightarrow f$, and $f \rightarrow b$, with state c remaining unchanged. A permutation can also be expressed as disjoint *cycles*. A cycle is basically an ordered list, which is represented as:

$$C = (e_1, e_2, \ldots, e_{n-1}, e_n). \tag{12}$$

The order of the elements describes the operation. For example, in Eq.(12), the cycle takes $e_1 \rightarrow e_2$, $e_2 \rightarrow e_3$, $\ldots$,$e_{n-1} \rightarrow e_n$, and finally $e_n \rightarrow e_1$. The number of elements in a cycle is called *length*. A cycle of length 1 is called a *trivial* cycle, which can be ignored as it does not change anything. A cycle of length 2 is called a *transposition*. Using this notation, the same permutation P shown in Eq.(11) can be written as

$$P = (a, d)(c)(b, e, f) = (a, d)(b, e, f). \tag{13}$$

As we can see, a simple quantum boolean gate like **CN** can be regarded as a permutation, because the probability amplitudes in the quantum state are manipulated in the same way. In other words, a quantum boolean logic gate can be expressed as a permutation, or cycles. For example, a **CN** gate is indicated by $P_{CN} = (10, 11)$, changing $10 \rightarrow 11$ and $11 \rightarrow 10$, leaving all other states unchanged.

In addition to permute the probability amplitude of each eigenstate, a qubit can be permuted as a whole. This is equivalent to reshuffling the quantum states for each of the qubits. Since a permutation can be decomposed into disjoint cycles, the implementation actually consists of executing cycles of various lengths in parallel. Because a cycle of

length 1 does not permute anything, no circuit is required for a trivial cycle. For a cycle of length 2, the transposition can be done by three **CN** gates, as shown in Fig.2(a).

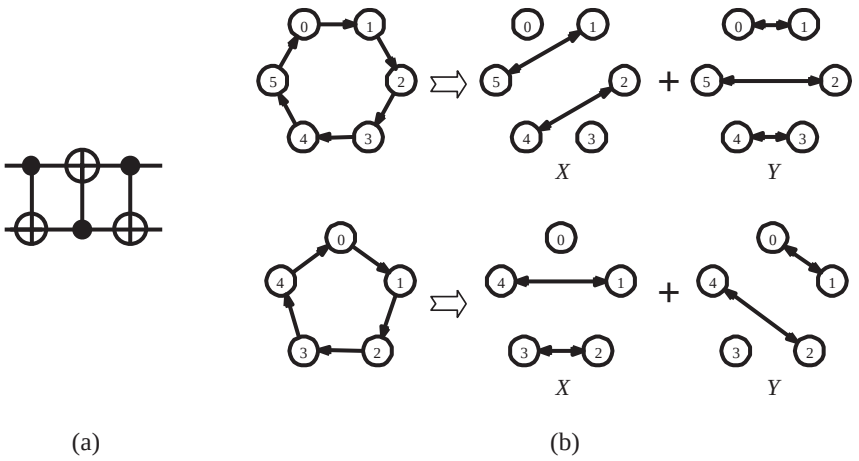

Figure 2: The circuit for (a) a transposition and (b) general cycles.

The circuit is described as follows. For a two-qubit system

$$|\psi, \phi\rangle = c_{00}|00\rangle + c_{01}|01\rangle + c_{10}|10\rangle + c_{11}|11\rangle, \tag{14}$$

the circuit transforms $|00\rangle \rightarrow |00\rangle$, $|01\rangle \rightarrow |10\rangle$, $|10\rangle \rightarrow |01\rangle$, and $|11\rangle \rightarrow |11\rangle$. This is equivalent to the permutation

$$P = (c_{00})(c_{01}, c_{10})(c_{11}). \tag{15}$$

Assuming the state of these two unentangled qubits are $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ and $|\phi\rangle = \gamma|0\rangle + \delta|1\rangle$, where $\alpha, \beta, \gamma, \delta \in C$ and $|\alpha|^2 + |\beta|^2 = |\gamma|^2 + |\delta|^2 = 1$, the joint state

$$|\psi\rangle \otimes |\phi\rangle = \alpha\gamma|00\rangle + \alpha\delta|01\rangle + \beta\gamma|10\rangle + \beta\delta|11\rangle \tag{16}$$

is transformed to

$$\begin{aligned} & \alpha\gamma|00\rangle + \beta\gamma|01\rangle + \alpha\delta|10\rangle + \beta\delta|11\rangle & (17)\\ = \; & (\gamma|0\rangle + \delta|1\rangle) \otimes (\alpha|0\rangle + \beta|1\rangle) & (18)\\ = \; & |\phi\rangle \otimes |\psi\rangle, & (19) \end{aligned}$$

which does the transposition. Note that once we have this basic function, we can build a switching network in the same way as a classical space switch. However, a more efficient implementation exists, as will be presented later in this paper.

For a general n-qubit ($n \geq 3$) cycle $C = (q_0, q_1, q_2, \cdots q_{n-1})$, it can be done by 6 layers of **CN** gates without ancillary qubits [13]. The quantum operations required to implement C are shown below.

For an even n ($n = 2m$, $m = 2, 3 \ldots$), we define the following non-overlapping qubit transpositions as:

$$X = (q_{m-1}, q_{m+1}) \cdots (q_2, q_{n-2})(q_1, q_{n-1}), \tag{20}$$

$$Y = (q_m, q_{m+1}) \cdots (q_2, q_{n-1})(q_1, q_0). \tag{21}$$

The cycle can be implemented using

$$U = YX. \tag{22}$$

On the other hand, for an odd n ($n = 2m+1$, $m = 1, 2, 3 \ldots$), we define the following non-overlapping qubit transpositions as:

$$X = (q_m, q_{m+1}) \cdots (q_2, q_{n-2})(q_1, q_{n-1}), \tag{23}$$

$$Y = (q_m, q_{m+2}) \cdots (q_2, q_{n-1})(q_1, q_0). \tag{24}$$

Note that if the subscript $m + 2 \geq n$ then $mod(m + 2, n)$ is used to avoid ambiguity. In the same way, the cycle can be implemented using

$$U = YX. \tag{25}$$

Two examples of $n = 5$ and $n = 6$ are shown in Fig.2(b).

Note that both X and Y consist of disjoint transpositions and can be executed in parallel using 3 layers of **CN** gates, as shown in Fig.2(a). As a result, each cycle and the whole permutation can be performed using 6 layers of **CN** gates. This achieves the constant time complexity of a qubit permutation. If auxiliary qubits are used, a cycle can be implemented using only 4 layers of **CN** gates [13].

In addition to permutation, qubit replication (**FANOUT**) is also an important and non-trivial operation. Qubit replication takes one bit as input and gives two copies of the same bit value as output. In the classical world, we can do this simply with a metallic contact, but it is well-known that quantum mechanics does not allow us to make an exact copy of an unknown qubit. This is called the quantum *non-cloning* theorem [14]. However, if the source qubit is in either $|0\rangle$ or $|1\rangle$, the quantum state can be replicated exactly using a **CN** gate. For example, if $|\psi\rangle$ is in either $|0\rangle$ or $|1\rangle$, replicating $|\psi\rangle$ to the qubit $|\phi\rangle = |0\rangle$ can be done simply by applying a **CN** gate with $|\psi\rangle$ as the control and $|\phi\rangle$ as the target, *i.e.* $CN(|\psi, 0\rangle)$. Moreover, since both $|\psi\rangle$ and $|\phi\rangle$ can be used

as the source qubits for further replication processes, the number of copies will increase exponentially, which allows C copies of the same quantum state being replicated using only $\log_2 C$ layers of **CN** gates, as shown in Fig.3. Note that the **CN** gates which have non-overlapping control and target qubits can be executed in parallel and are grouped into one layer.

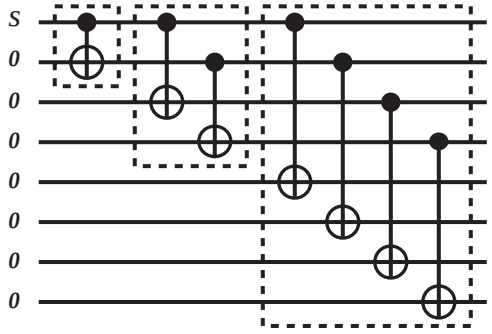

Figure 3: An example of qubit replication from $s = |0\rangle$ or $|1\rangle$ to multiple targets.

3 Classical Digital Switching Techniques

In classical digital communication, switching is needed in order not to build a fully-meshed transmission network. Generally, digital switching technologies fall under two broad categories: *circuit switching* or *packet switching*. In this section, we briefly introduce these two switching paradigms and describe various implementations that can be employed to implement the switching function. We also define the connection digraph which can be used to illustrate the switching operation at a given time.

3.1 Digital Switching Networks

In circuit switching, a dedicated path or time slot is reserved for an end-to-end bandwidth demand. The connection is established at the time of call set-up and released when the call is torn down. The function of the switching module is to transfer a particular time slot in the input port to a time slot in the output port. Assuming A (time slot $S0$ of port $P1$) and B (time slot $S2$ of port $P2$) are making two-way communication via a 4×4 digital switch, as shown in Fig.4(a). For the connection from A to B, the switching module transfers the data x from $S0$ of $P1$ to $S2$ of $P2$. Similarly, for the connection from B to A, it transfers the data y from $S2$ of $P2$ to $S0$ of $P1$. These operations complete the data exchange between A and B.

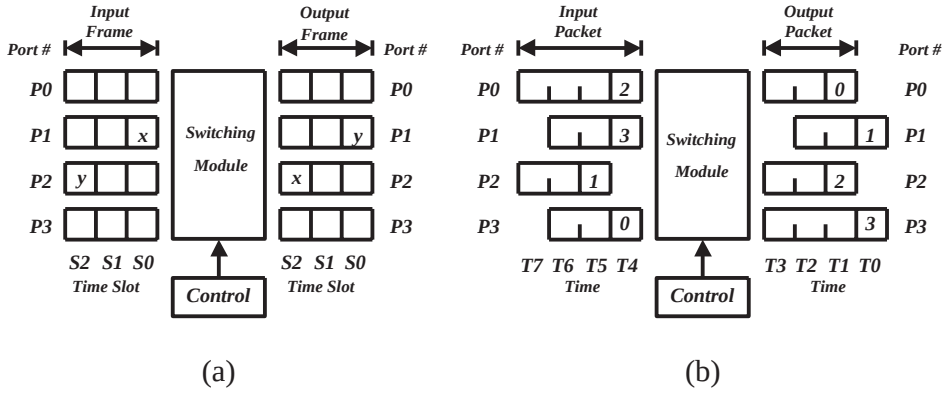

Figure 4: Examples of (a) circuit switching and (b) packet switching.

Packet switching is more sophisticated than circuit switching. Modern packet switching networks take packets that share the same transmission line as input. A packet can have either a fixed or variable length with a limited maximum size. When a packet arrives at a node, it is stored first and then forwarded to the desired node according to its header as shown in Fig.4(b). For example, assume each of the packets in Fig.4(b) has the destination port number as indicated in the header of the packet. The switching module at time $T5$ needs to switch the data from input port $P0$, $P1$, $P2$, and $P3$ to output port $P2$, $P3$, $P1$, and $P0$ respectively.

Although significant differences such as *data dependency* and *output contention* exist between circuit switching and packet switching, they still have similarities. In both circuit switching and packet switching, the control block needs to specify the switching configuration for each individual time slot, so the data in that particular time slot can be switched correctly. The configuration describes how the I/O ports should be switched at a given time. The actual switching operation depends on which switching technique is used. There are many switching techniques used today. Some of the basic switching techniques are described in the following section.

In the field of classical digital switching, various techniques have been used to switch the input data to the corresponding output port. For example, data can be switched in the space domain, the time domain, or the wavelength domain, etc. If the data is switched in the space domain, *i.e.* space division switching, usually a physical path or a dedicated time slot is reserved to establish the connection. For example, in the crossbar architecture, a rectangular array of cross-points serve as a simple space switching architecture. Every output port can be reached by every input port in a non-blocking way by closing a single cross-point. A more sophisticated space division switch utilizes multiple stages of rectangular arrays is shown in Fig.5(a). A connection is established by closing proper

cross-points to select a path from the inlet to the outlet [15].

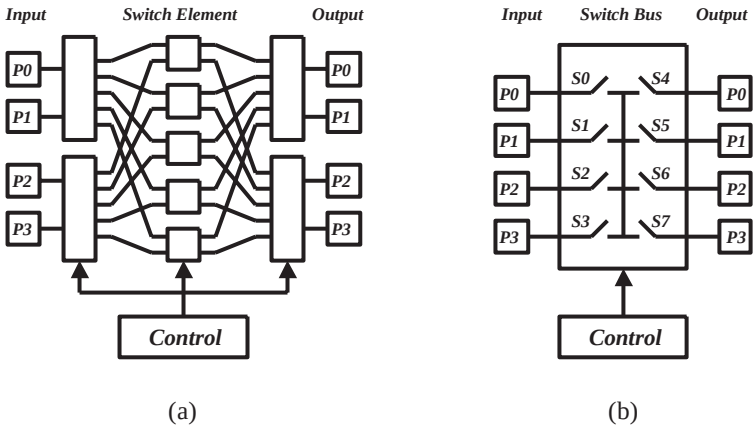

Figure 5: Examples of classical digital switching techniques.

A device that switches the data in the time domain is called a time division switch. Time division technology is widely used in modern digital communication. In a time division switch, connections are established in a time-sharing manner, so a connection occupies the resources for only a short duration of time. For example, in Fig.5(b), the connection from the inlet $P1$ to the outlet $P3$ is established by closing switch $S1$ and $S7$. This process is executed for each of the connections in a cyclic way to achieve switching functionality. Primarily owing to the low cost of semiconductor devices, the implementation of a time division switch is usually done by using digital memory. Data received over an incoming port is written into the memory, the switching is accomplished by reading out the individual bits in the desired time slot, which is equivalent to connecting the inlet to the outlet for data transfer.

3.2 Connection Digraphs

Before we describe how digital switching can be done in the quantum domain, we define a *connection digraph* as follows:

Definition 1: Given an $n \times n$ switch, the connection digraph at time t, $G^t = \{V, E^t\}$, is a digraph such that

1. Each $v_i \in V (i = 0, 1, \ldots n-1)$ represents an I/O port.
2. $\overrightarrow{v_m v_n} \in E^t$ if and only if a connection exists from the input port v_m to the output port v_n at time t. ■

In a connection digraph, each node represents an I/O port, a directed edge $\overrightarrow{v_m v_n}$ is used to describe a connection when the connection from input port v_m to output port v_n is active. The digraph describes the connection status of the switch at a given time, and is called the connection digraph at time t. Note that the directed edge $\overrightarrow{v_m v_n}$ denotes only a one-way data path. For a point-to-point two-way communication between v_m and v_n, both $\overrightarrow{v_m v_n}$ and $\overrightarrow{v_n v_m}$ have to be used. Obviously, due to the connection set-up and torn-down processes, the connection digraph is a function of time.

Depending on the status of the switch, the topology of a connection digraph varies. In a general digraph, it is possible that a node has multiple predecessors and multiple successors. However, when there is no output contention or the problem is solved elsewhere, each node will have at most one predecessor. As to the number of successors, it depends on the type of the connection. In a multicast connection, the source node has multiple successors, while in a unicast connection, only a single successor is possible. In the following sections, we will discuss the connection digraph based on this model and show that any connection digraph actually consists of a set of basic topologies as disjoint sub-digraphs. These basic topologies are defined as follows:

Definition 2: Given a digraph $G = (V, E)$ with only one node, *i.e.* $V = \{v\}$. G is called a *null node* if $E = \emptyset$. Otherwise G is called a *loopback* when $E = \{\overrightarrow{vv}\}$. ■

In a connection digraph, a null node without predecessor and successor means there is neither input traffic coming from that port nor output traffic going to that port. For a port without incoming traffic, we assume the stuff bits are all 0's. However, a single node with a directed edge to itself means the input traffic goes back to the same port. This trivial cycle effectively denotes a loopback. A loopback G^L can be made from a null node G^N simply by linking the null node to itself. G^L is called the extension loopback of G^N, denoted by $E(G^N)$. An example consists of null nodes and loopbacks is shown in Fig.6(a). The numbers in the boxes represents the destination port numbers. An 'X' represents no input traffic. Its corresponding connection digraph is depicted in Fig.6(b).

Definition 3: Given a connected digraph $G = (V, E)$ with n ($n \geq 2$) nodes. G is called a *queue* if

1. there exists one and only one *head* $v_h \in V$, such that for each $v_i \in V$, $\overrightarrow{v_i v_h} \notin E$.
2. there exists one and only one *tail* $v_t \in V$, such that for each $v_i \in V$, $\overrightarrow{v_t v_i} \notin E$.
3. for each $v_i \in V (i \neq t)$, there exists one and only one v_j, such that $\overrightarrow{v_i v_j} \in E$. ■

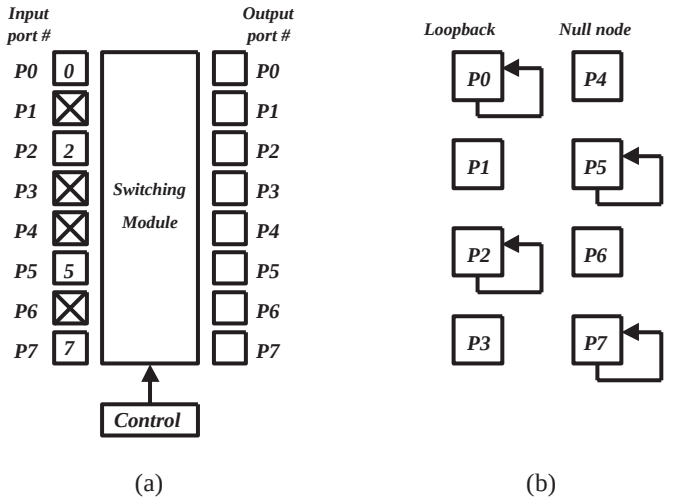

Figure 6: A connection digraph with null nodes and loopbacks.

A queue can be represented as a linear array from the head v_h to the tail v_t, and is denoted as $[v_h, v_1, v_2, \ldots, v_{n-2}, v_t]$. This notation means the connection at a given time includes $\overrightarrow{v_h v_1}$, $\overrightarrow{v_1 v_2}$, ..., and $\overrightarrow{v_{n-2}, v_t}$. Note that there is no input traffic coming from v_t and no output traffic going to v_h. An example of a queue connection is shown in Fig.7(a), with its connection digraph $G^Q = [P2, P4, P3, P7, P5, P6, P0, P1]$ depicted in Fig.7(b). Each connection in a queue is apparently a unicast connection, because there is at most one outgoing arrow from each node.

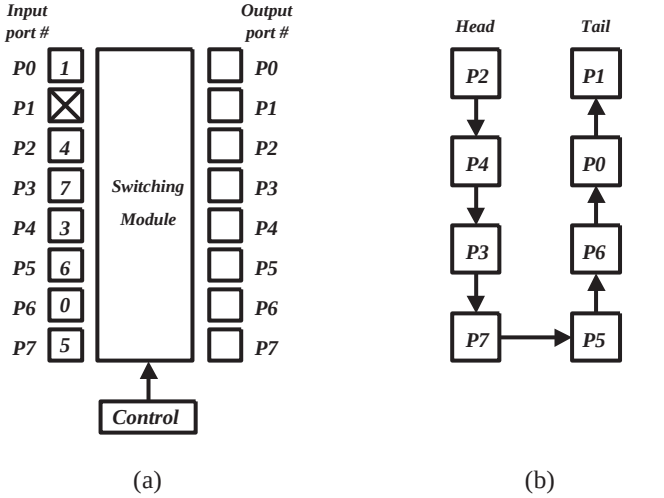

Figure 7: An example of a queue connection and its connection digraph.

Connecting the tail to the head of a queue forms a *cycle*, which is defined as follows:

Definition 4: Given a connected digraph $G = (V, E)$ with n ($n \geq 2$) nodes, G is called a cycle if

1. for each $v_i \in V$, there exists one and only one v_j, such that $\overrightarrow{v_j v_i} \in E$.
2. for each $v_i \in V$, there exists one and only one v_k, such that $\overrightarrow{v_i v_k} \in E$. ■

Using the same notation, a cycle connection is represented as $(v_0, v_1, v_2, \ldots, v_{n-2}, v_{n-1})$. This means the connection at a given time includes $\overrightarrow{v_0 v_1}$, $\overrightarrow{v_1 v_2}$, $\ldots$,$\overrightarrow{v_{n-2} v_{n-1}}$, and $\overrightarrow{v_{n-1}, v_0}$. In the case of a cycle, each port has its input as well as output. As described earlier, the tail and head of a queue G^Q can be connected to form a cycle G^C. G^C is called the extension cycle of G^Q, denoted by $E(G^Q)$. An example of a cycle connection is shown in Fig.8(a), with its connection digraph $G^C = (P2, P4, P3, P7, P5, P6, P0, P1)$ depicted in Fig.8(b).

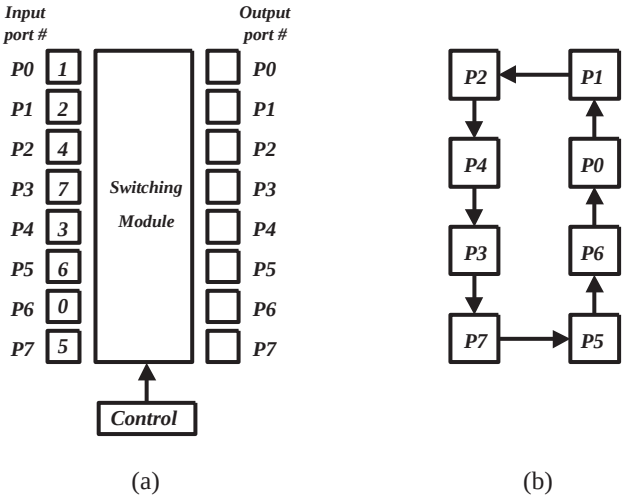

Figure 8: An example of a cycle connection and its connection digraph.

In order to describe a multicast connection, we define the following connection digraphs:

Definition 5: Given a connected digraph $G = (V, E)$ with n ($n \geq 2$) nodes, G is called a *tree* if

1. there exists one and only one *root* $v_r \in V$, such that for each $v_i \in V$, $\overrightarrow{v_i v_r} \notin E$.

2. there exists a collection of nodes L called leaves, such that for each $v_l \in L$ and $v_i \in V$, $\overrightarrow{v_l v_i} \notin E$.

3. for each $v_i \in V - L$, there exists at least one v_j, such that $\overrightarrow{v_i v_j} \in E$. ■

The nodes in a tree can be divided into three categories: root, internal nodes, and leaves. For the root, the output data is directed to possibly multiple output ports, but no data goes to the root. However, all leaves receive data without generating traffic. All internal nodes have exactly one predecessor and at least one successor. A tree can be represented as a concatenation of queues like $G^T = [v_h^0, \ldots v_t^0][v_h^1, \ldots v_t^1] \ldots [v_h^n, \ldots v_t^n]$, with v_h^0 be the root and each of the v_h^n $(n \geq 1)$ be the tail of one of the previous queues. An example of a tree connection is shown in Fig.9(a). If there are multiple numbers in a box, they represent a multicast connection. Its corresponding connection digraph $G^T = [P1][P1, P3][P1, P6, P4][P3, P5][P3, P7][P4, P0][P4, P2]$ is depicted in Fig.9(b). Note that a queue is a special case of trees, with each node having only one successor.

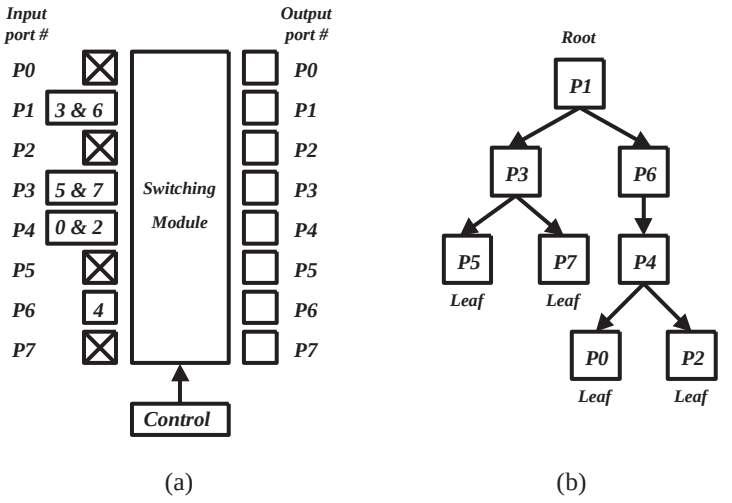

Figure 9: An example of a tree connection and its connection digraph.

Connecting any leaf to the root of a tree forms a *forest*, which is defined as follows:

Definition 6: Given a connected digraph $G = (V, E)$ with n nodes $(n \geq 2)$, G is called a forest if

1. there is one and only one cycle $G^C = (V^C, E^C)$ exists as a sub-digraph of G.

2. let $G' = \{\overrightarrow{v_i v_j} \mid v_i \in V^C, \overrightarrow{v_i v_j} \in E, \overrightarrow{v_i v_j} \notin E^C\}$. $G - G'$ contains the cycle G^C and a collection of disjointed null nodes, queues, and/or trees.

3. each v_j is either one of the null nodes, the head of a queue, or the root of a tree in $G - G'$. ■

A forest basically contains one and only one cycle $G^C = (V^C, E^C)$ as a sub-digraph, with some of its nodes linked to either a null node, the head of a queue, or the root of a tree. Following this structure, a forest can be represented by $G^F = \{G^C, G^1, G^2, G^3 \cdots\}$, where G^1, G^2, G^3, ... be either a null node, a queue, or a tree. A forest can be extended from a tree by connecting any leaf to the root. A forest G_l^F formed by connecting the leaf l with the root of G^T is called the extension forest of G^T, denoted by $E_l(G^T)$. An example of a forest connection is shown in Fig.10(a), with its connection digraph $G^F = \{(P4, P1, P3, P6), [P3][P3, P5][P3, P7], [P4][P4, P2][P4, P6]\}$ depicted in Fig.10(b).

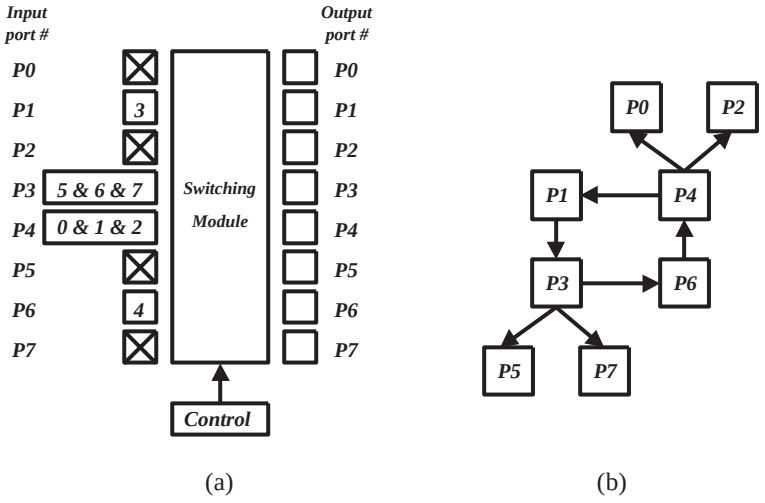

Figure 10: An example of a forest connection and its connection digraph.

Since each node in a unicast connection has at most one successor, a unicast connection digraph only consists of disjoint null nodes, loopbacks, queues, and/or cycles as sub-digraphs. However, a multicast connection switches the data from one node to multiple successors, so a multicast connection digraph consists of disjoint null nodes, loopbacks, queues, cycles, trees, and/or forests as sub-digraphs. Based on these results, we describe the architecture of quantum switching and show how it can be used to implement a connection digraph in the next section.

4 Switching in the Quantum Domain

4.1 Principle of Digital Quantum Switch

The proposed architecture for building a digital *quantum switch* is depicted in Fig.11. To switch classical digital data in the quantum domain, first we have to convert the classical data into qubits. For example, in a quantum switch with optical I/O ports, an optical to quantum converter (O/Q) is used to convert optical input into qubits. In an O/Q, '0' is converted into $|0\rangle$ and '1' is converted into $|1\rangle$. This can be done by exciting an electron using a light pulse of a certain frequency. All qubits are then permuted (*i.e.* switched) by the unitary operations under the supervision of the control subsystem. After the permutation, all qubits are converted back into their optical form by a quantum to optical converter (Q/O). This can be done by measuring the qubits to recover the original classical information.

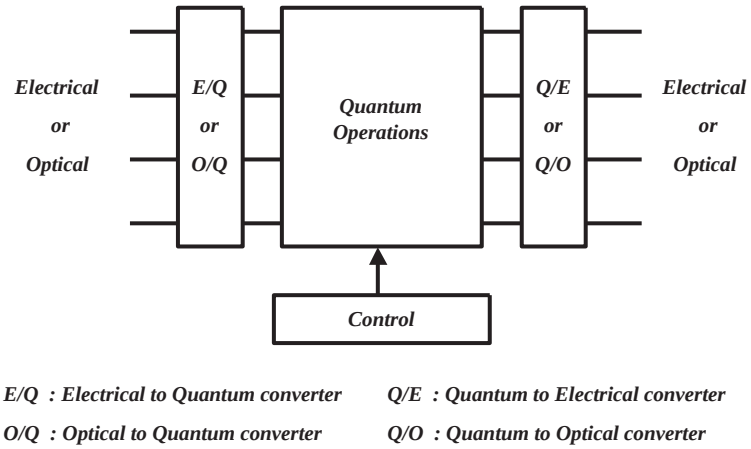

Figure 11: The architecture of a digital quantum switch.

4.2 Connection Digraph Implementation

In this section, we show how a connection digraph can be implemented using **CN** gates. First we describe the connection digraph transformation guideline, then we demonstrate how this guideline can be used to implement a connection digraph. Both unicasting and multicasting will be covered in detail.

4.2.1 Guideline for implementing a Connection Digraph

As described earlier, due to the nature of the connection, unicasting and multicasting have different connection digraphs. The digraph of a unicast connection has a collection

of disjointed null nodes, loopbacks, queues, and/or cycles as sub-digraphs. However, in the digraph of a multicast connection, sub-digraphs like trees and forests are possible. As a matter of fact, these topologies are inter-related. This is shown in Fig.12 and summarized as follows:

1. A null node can be regarded as a special case of a queue, denoted by the arrow S1.
2. A queue can be regarded as a special case of a tree, denoted by the arrow S2.
3. A loopback can be regarded as a special case of a cycle, denoted by the arrow S3.
4. A cycle can be regarded as a special case of a forest, denoted by the arrow S4.

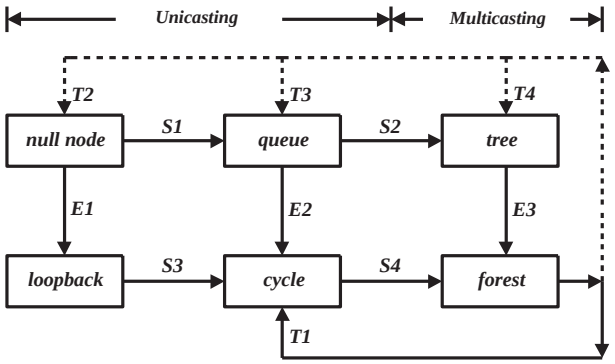

Figure 12: Inter-related connection topologies.

Of course, the binary relation "*is a special case of*" is transitive, so a null node and a loopback are special cases of tree and forest respectively. Fig.12 also shows the binary relation "*can be extended to*" as follows:

1. A null node G^N can be extended to a loopback $G^L = E(G^N)$, denoted by the process E1.
2. A queue G^Q can be extended to a cycle $G^C = E(G^Q)$, denoted by the process E2.
3. A tree G^T can be extended to a forest $G^F = E_l(G^T)$, denoted by the process E3.

Note that the process of extension only transfers the incoming data from an idle inlet (all 0's) to an outlet which has no outgoing traffic, this does not change the switching function.

The first step of our guideline for implementing a connection digraph is to transform each disjointed sub-digraph into loopbacks and/or cycles. Since no circuit is needed to implement a loopback and only 6 layers of **CN** gates are sufficient to implement a cycle,

the switching can be done efficiently. Some of these transformations are straightforward. For example, following E1, a null node G^N can be extended to a loopback $G^L = E(G^N)$. Also, following E2, a queue G^Q can be extended to a cycle $G^C = E(G^Q)$. However, for a tree or a forest, "*cycle extraction*" and "*link recovery*" have to be used. The process of cycle extraction and link recovery are described as follows.

Cycle Extraction: A forest basically contains one and only one cycle $G^C = (V^C, E^C)$ as a sub-digraph with a subset of V^C linked to either a null node, the head of a queue, or the root of a tree. The process of cycle extraction detaches all the null nodes, queues, and trees from the cycle by cutting all the edges in $E = \{\overrightarrow{v_i v_j} \mid v_i \in V^C, v_j \notin V^C\}$, as shown in Fig.13(a). This will transform a forest into one cycle (arrow T1 in Fig.12) and a collection of null nodes, queues, and/or trees (arrow T2, T3, and T4 respectively). Each of the null nodes and queues can further be transformed into loopbacks and cycles via process E1 and E2. If there are still any trees in the remaining digraph, extensions can be made again to transform the trees into forests (process E3) and the procedure of cycle extraction can be applied recursively (arrow T1, T2, T3, and T4) until no trees are left. This procedure eventually transforms a forest into loopbacks and/or cycles, so that the permutation can be implemented using 6 layers of **CN** gates in parallel.

Link Recovery: After each cycle has been implemented, the links that had been cut must be recovered. That is, for each $\overrightarrow{v_i v_j} \in E^C$, if $\overrightarrow{v_i v_k} \in E$ but $\overrightarrow{v_i v_k} \notin E^C$, v_j must be replicated to v_k, as shown in Fig.13(b). Since there will be at most $n-2$ such k's in a multicast connection digraph, in the worst case the replication can be done by $\log_2 n$ layers of **CN** gate. This completes the implementation of a forest. For a tree, it can be extended to a forest via process E3 and then follow the algorithm to do further reduction in the same way.

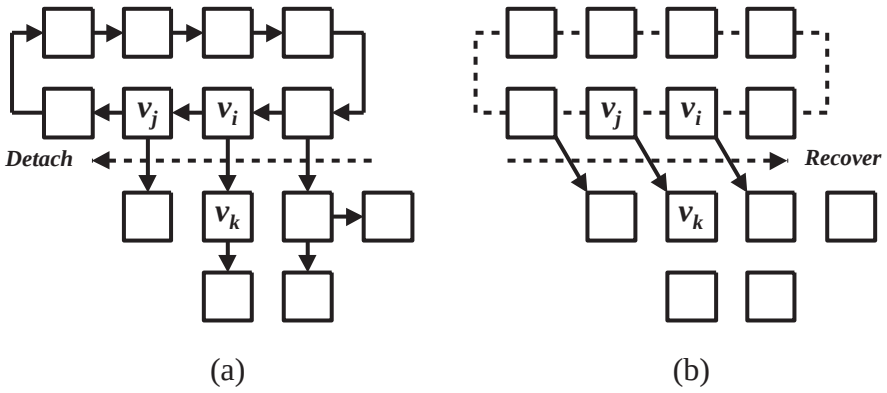

Figure 13: The process of (a) cycle extraction and (b) link recovery.

4.2.2 Unicast Connection Digraph

Following the guideline described above, in this section we show how a unicast connection digraph can be implemented with a time complexity of $O(1)$ and a space complexity of $O(n)$. A typical unicast connection status at a given time is shown by the solid arrows in Fig.14(a). The switching module needs to perform two connection sub-digraphs:

$$G^C = (q_3, q_4, q_6, q_7, q_5), \tag{26}$$

$$G^Q = [q_0, q_1, q_2]. \tag{27}$$

These can be done by first extending G^Q to $G^{C'} = (q_0, q_1, q_2)$, as shown by the dash link in Fig.14(a), and then implement G^C and $G^{C'}$ using 6 layers of **CN** gates. As described previously, the sub-digraph $G^C = (q_3, q_4, q_6, q_7, q_5)$ can be done by first applying

$$X = (q_6, q_7)(q_4, q_5) \tag{28}$$

and then

$$Y = (q_6, q_5)(q_4, q_3). \tag{29}$$

The transposition (q_4, q_5) is done by

$$(q_4, q_5) = CN(q_4, q_5) \cdot CN(q_5, q_4) \cdot CN(q_4, q_5), \tag{30}$$

as shown by block B in Fig.14(b). In the same way, (q_6, q_7), (q_4, q_3), (q_6, q_5) are done by blocks C, E, F respectively. Similarly, the implementation of $G^{C'} = (q_0, q_1, q_2)$ can be done by first applying $X = (q_1, q_2)$ and then $Y = (q_1, q_0)$. These are implemented as blocks A and D in Fig.14(b).

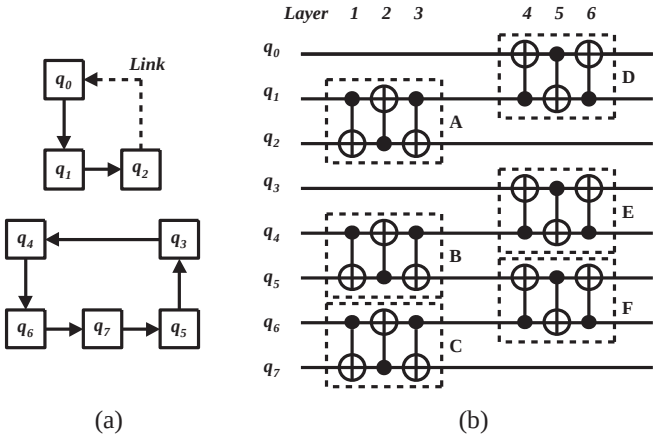

Figure 14: (a) A unicast connection digraph, and (b) its quantum circuits

Note that, independent of the switch size n, the whole circuit can be completed in 6 layers of **CN** gates over n qubits. This achieves a time complexity of $O(1)$ and a space complexity of $O(n)$.

4.2.3 Multicast Connection Digraph

In classical packet switching, the input packets are usually buffered in the memory, multicasting can be easily achieved by reading the packet once and writing the same packet to multiple destinations. If the switching is done in the quantum domain, multicasting can be done by replicating the input qubit to multiple destination qubits. A typical multicasting configuration is shown in Fig.15(a). In this example, the switching module needs to perform the following connection digraph:

$$G^T = [q_0, q_1][q_1, q_4][q_1, q_3][q_3, q_5, q_2][q_3, q_6, q_7]. \tag{31}$$

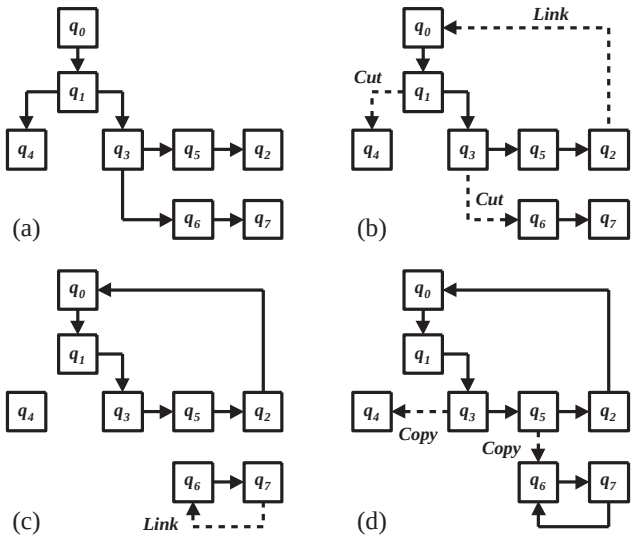

Figure 15: Procedures for implementing a multicast connection digraph.

Following the guideline, each of the steps is shown below:

1. The tree G^T can be extended to a forest by linking any leaf, say q_2, to q_0. The cycle extraction procedure is then performed to cut $\overrightarrow{q_1 q_4}$ and $\overrightarrow{q_3 q_6}$ down. The result is shown in Fig.15(b).

2. The extension and cycle extraction processes are recursively applied to $[q_6, q_7]$ until no tree is left, as shown in Fig.15(c).

3. Each of the disjointed sub-digraphs can be implemented in parallel. The sub-digraph $G^C = (q_0, q_1, q_3, q_5, q_2)$ can be done by first applying $X = (q_1, q_2)(q_3, q_5)$ and then $Y = (q_1, q_0)(q_3, q_2)$, while $G^{C'} = (q_6, q_7)$ can be implemented directly, as shown by blocks A, B, D, E, and C in Fig.16.

4. Each of the disconnected edges has to be recovered, so q_3 needs to be replicated to q_4, and q_5 needs to be replicated to q_6, as shown in Fig.15(d). These can be done by blocks F and G in Fig.16.

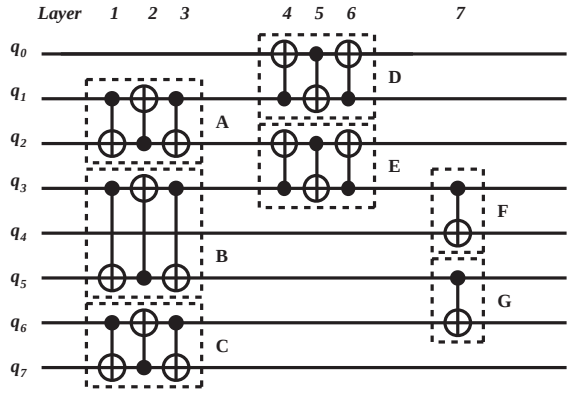

Figure 16: Quantum circuits for a multicast connection digraph.

In general, the total number of layers for implementing a multicast connection digraph is $6 + \lceil \log_2(r+1) \rceil$, where r is the maximum number of $\overrightarrow{v_j v_k}$ $(k = 1, 2, \ldots r)$ that are to be recovered. In the worst case, when one inlet is broadcast to all other $n-1$ outlets, the whole connection digraph can be done in $O(\log_2 n)$ layers of **CN** gates over n qubits. This results in a time complexity of $O(\log_2 n)$ and a space complexity of $O(n)$.

4.3 Advantages of Quantum Switching

The advantages of performing digital switching in the quantum domain are summarized as follows. First, switching in the quantum domain is *strict-sense non-blocking*. A switch is called strict-sense non-blocking if the network can always connect each idle inlet to an arbitrary idle outlet independent of the current network permutation [10]. Note that switching in the space domain is not always non-blocking. Sometimes, the required data path can not be established even if the output port is available. It has been shown that for an $n \times n$ network in Fig.5(a)) to be non-blocking, there must be at least $2n-1$ modules in the middle stage [15]. However, switching in the quantum domain is actually a unitary transformation, which is always possible. This results in the fact that a quantum switch is non-blocking in the strict sense.

Second, it takes only n qubits to build a quantum switch, the space complexity is $O(n)$ in terms of the number of qubits. The problem of space complexity is an important issue in the classical space switching. To make a classical space switch non-blocking, a certain number of modules in the middle stage have to be used to allocate a physical path

for each connection, so the number of cross-points increases with the size of the switch. For example, with optimal grouping, the minimum number of cross-points for the switch shown in Fig.5(a) is $N_{min} = 4n(\sqrt{2n} - 1)$, where n is the total number of inlets/outlets [15]. However, an $n \times n$ quantum switch uses only n qubits as the basis to perform the switching, which is a reasonable resource consumption.

Third, quantum switching is scalable in terms of time complexity. In a classical time switch, usually the bottleneck is the speed of the switching device. Because when the throughput increases, the time duration for switching a particular bit of data decreases. For example, in a memory switch with throughput T, the memory speed must be at least $1/2T$ to allow one read and one write operation to be performed. However, in the quantum switching, the time complexity is not sensitive to the throughput. A high throughput quantum switch can be achieved simply by increasing the number of I/O ports, which only induces a reasonable amount ($O(n)$) of space consumption. However, even in the worst case scenario, the throughput gain still outweights the time penalty in a classical time domain switch ($O(n)$ v.s. $O(\log_2 n)$).

5 Conclusions

Networks are rapidly growing due to increased number of users and rising demands for bandwidth-intensive services. To support such a huge traffic volume, a wide range of different technologies are being proposed as the core of a high performance switch. In this paper, an architecture of digital quantum switching is presented. The proposed mechanism allows digital data to be switched using a series of quantum operations. The procedures of how to implement unicast and multicast connections are discussed in detail. In terms of the blocking rate, this architecture is strict-sense non-blocking. From a complexity point of view, the space complexity grows only linearly with the number of I/O ports, and the time complexity is constant for unicasting and logarithmic for multicasting. This architecture is suitable for deploying high throughput switching devices so that high bandwidth demand can be met.

References

[1] P. Benioff, "The computer as a physical system: a microscopic quantum mechanical hamiltonian model of computers as represented by turing machines," *J. Stat. Phys.*, vol. 22(5), pp. 563-591, 1980.

[2] R. Feynman, "Simulating physics with computers," *Int. J. Theor. Phys.*, vol. 21, pp. 467-488, 1982.

[3] D. Deutsch, "Quantum theory, the Church-Turing principle and the universal quantum computer," *Proc. Roy. Soc. Lond. A*, vol. 400, pp. 97-117, 1985.

[4] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proc. of the 35th Annual IEEE Symposium on the Foundations of Computer Science*, 1994, pp. 124-134.

[5] L. Grover, "A fast quantum mechanical algorithm for database search," in *Proc. of the 28th Annual ACM Symposium on the Theory of Computing*, 1996, pp. 212-219.

[6] C. Bennett and G. Brassard, "Quantum Cryptography: Public Key Distribution and Coin Tossing," in *Proc. of IEEE International Conference on Computers Systems and Signal Processing*, 1984, pp. 175-179.

[7] R. Jozsa, D. Abrams, J. Dowling, C. Williams. (2000, Apr.). Quantum Clock Synchronization Based on Shared Prior Entanglement. [Online]. Available: http://www.arXive.org/quant-ph/0004105/.

[8] I. Chuang. (2000, May.). Quantum algorithm for distributed clock synchronization. [Online]. Available: http://www.arXive.org/quant-ph/0005092/.

[9] I. M. Tsai and S. Y. Kuo, "Quantum boolean circuit construction and layout under locality constraint," in *Proc. of the 1st IEEE Conference on Nanotechnology*, 2001, pp. 111-116.

[10] A. Pattavina. *Switching Theory*, West Sussex, UK: John Wiley & Sons, 1998, pp.54.

[11] D. DiVincenzo, "Two-bit gates are universal for quantum computation," *Phys. Rev. A*, vol. 51(2), pp. 1015-1022, 1995.

[12] A. Barenco, "A universal two-bit gate for quantum computation," *Proc. Roy. Soc. Lond. A*, vol. 449, pp. 679-683, 1995.

[13] C. Moore and M. Nilsson. (1998, Aug.). Parallel quantum computation and quantum codes. [Online]. Available: http://www.arXive.org/quant-ph/9808027/.

[14] W. Wootters and W. Zurek, "A single quantum cannot be cloned," *Nature*, vol. 299, pp. 802-803, 1982.

[15] C. Clos, "A study of nonblocking switching networks," *BSTJ.*, vol. 32, no. 2, pp. 406-424, 1953.